Cybernetic Ducks

Statistics 101C Lecture 4 Midterm Project Report

Ivan Escusa, Oscar Monroy, Zoe Wang,
and Christine Marie Castle

2020-11-09

1 Introduction

Cancer is a group of diseases involving abnormal cell growth, most of which are caused by genetic mutations. Oncogenes (OG) are genes that aid in cell growth. Tumor suppressor genes (TSG) are genes that can slow cell division, repair DNA mistakes, or direct programmed cell death. The accumulation of mutations of either of these genes may result in uncontrolled cell growth and lead to cancer. Thus, it is imperative to discover cancer driver genes such as OG's and TSG's for cancer prevention, diagnosis, and treatment.

In the past few decades, scientists have identified about 250 OG's, 250 TSGs, and about 4000 neutral genes (NG), i.e. neither OG's nor TSG's, out of the 20,000 genes in the whole human genome. In recent years, many large sequencing projects, launched to catalogue genetic mutations responsible for cancer, have generated unprecedentedly large data resources. We will use statistical methods that we have learned in this course on a comprehensive gene feature data set to identify OG's and TSG's.

2 Methodology

2.1 Preprocessing

All of our cleaning was done using the `tidyverse` and `leaps` packages in `R`. To clean the raw data set provided to us, we first checked for missing values and duplicated rows in order to replace or remove them. We found one duplicate row, which was removed, but no missing values. Next, we looked for variables with non-numeric values and luckily found that this data set has only numeric values. In particular, the `id` and `class` variables only consisted of integer values. After these preliminary checks, we removed the outliers, defined as observations with values more than ten standard deviations away from their respective mean, from the data set. Outliers were found to make up less than 3% of the rows. Finally, to visually interpret differences between classes, box plots were made for each variable, split by the three classes of genes.

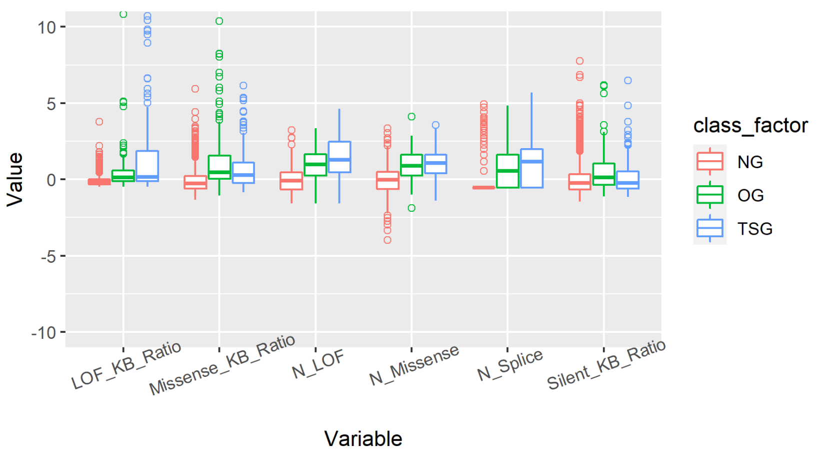

Figure 1: Sample of box plots used for initial variable selection

2.2 Statistical Model

For initial variable selection, we created box plots for each of the 97 potential predictors, split by the three classes of genes; we standardized the variables before plotting. We then used these box plots to assess which variables had the most pronounced differences between the three gene classes. Our motivation for this was to find predictors which had the greatest contrast between classes, as we believed those would be the most effective and efficient in differentiating the three gene classes. Additional consideration was given to the variables which showed greater potential in correctly classifying the oncogenes and tumor suppressor genes. For example, the variable `N_LOF`, which represents the number of LoF (nonsense, frameshift) mutations, was initially selected based on the considerable differences in range and median among the three classes as shown in the sample of box plots in Figure 1.

Our model utilizes Quadratic Discriminant Analysis (QDA) to make the predictions among the three classes. We decided on the QDA method because we desired the flexibility that QDA provides as opposed to Linear Discriminant Analysis (LDA) due to the size of our data set. K-Nearest Neighbors (KNN) was also attempted, however it was consistently outperformed by QDA regardless of the value of K selected. We assessed model performance with 5-fold cross-validation. We did not consider Logistic Regression because it is not commonly used when there more than two response classes (ISLR p.137).

Next, we created an initial model for predicting the class of an unidentified gene using the predictors we identified from the box plots. To refine our predictors after the initial variable selection, we created two functions, `removal_kfold_qda` and `addition_kfold_qda`, that utilize for loops to cycle through variables, removing or adding them one at a time respectively. Within these loops, we used 5-fold cross-validation and calculated the average error rate, storing them in a list; this process was repeated 15 times to ensure the error rate changed significantly enough to justify

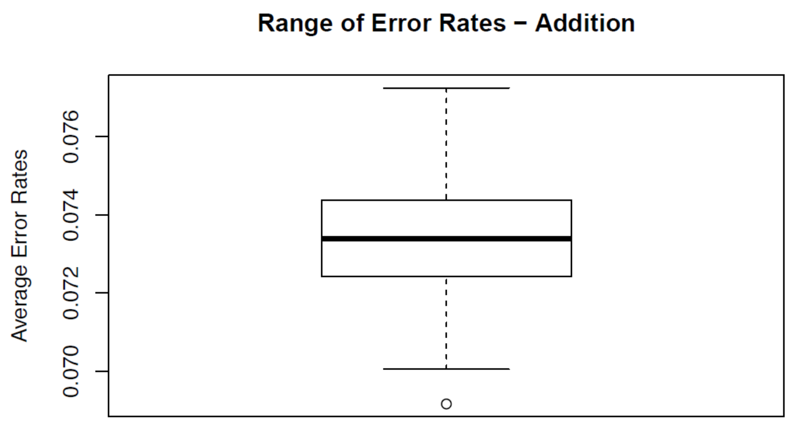

Figure 2: Box plot of average error rates produced with the removal loop

adding or removing one or more variables. With that, we generated summary statistics and box plots to provide visualizations of the range and to determine if there was enough change to the predictions' viability.

Figure 2 shows the box plot of the removal loop. The results display an outlier at the bottom, which indicates to us to remove the variable representing the outlier as a predictor for the model. In this way, we can lower the overall average error rate of our predictions to approximately 6.9%. After further refining based on this process and testing through submissions to Kaggle, we settled on our final model, which features 39 of the given 97 variables as predictors.

3 Results

Our best evaluation metric was a Kaggle score of 0.76644.

4 General Conclusions

One of the reasons QDA may have worked well for this problem is because we had a large set of variables to select from. The large number of observations in the data set also mitigated the increase in variance, which allowed for more flexibility and thus a lower bias in our model. On the other hand, the model may not have worked as well as we hoped because the increase in variance was still significant enough to cause overfitting to the training data. Our model may also have suffered if the underlying assumption of QDA, that the predictors are approximately normally distributed in each of the classes, was violated. Another downfall of our model is that the methodologies used to get to the final predictions, namely the loops, are computationally expensive and required several minutes to run.

Statement of Contributions

All four members of the Cybernetic Ducks: Ivan Escusa, Oscar Monroy, Zoe Wang, and Christine Marie Castle contributed to submitting data for the Kaggle Competition as well as providing the write-up and report for our process with the best model as evaluated with Kaggle. Our team made a total of 33 submissions for this project, 25 of which were successful. For this project, each of the members contributed to different parts of the Kaggle Competition. Christine Marie Castle cleaned up the data, created the box plots, submitted one Kaggle prediction that didn't make it above the benchmark, and typeset this report. Once the data was cleaned, Oscar Monroy contributed to submitting 18 different sets of predictions and was able to improve the team's best predictions after his 2nd, 7th, 8th, 12th, and 15th submissions. Our best prediction was on his 15th successful submission, the team's 22nd successful submission, and the 29th overall prediction including ones that had errors. Ivan Escusa also contributed three submissions to the Kaggle competition, all of which were barely above the benchmark of 0.5983 and weren't effective enough. Zoe Wang contributed three submissions to Kaggle, one with a score of 0.66, which was the highest score at the time, but the other submissions were below the benchmark. All four members also contributed to the visualizations and the appendix with all of the code in the finalization of this report.

Appendix

```r
library(caret)
library(ggplot2)
library(leaps)
library(MASS)
library(tidyverse)

### PREPROCESSING ###

training <- read.csv("training.csv")
testing <- read.csv("test.csv")

# Calculate Z scores of training data (except id and class)
sc_training <- training %>%
  select(-id, -class) %>%
  mutate_all(scale)
# Select rows that contain Z scores less than -10 or greater than 10
outliers_inds <- unique(which(abs(sc_training) > 10, arr.ind = TRUE)[, "row"])
# Remove outliers from training data
training1 <- training[-outliers_inds,]
training1 <- training1 %>%
  # Remove duplicated rows
  distinct_at(vars(!id), .keep_all = TRUE) %>%
  # Convert class from numeric to factor
  mutate(class = factor(class))

### BOX PLOTS ###

# Factor levels of class labeled with corresponding gene class
training0 <- training1 %>%
  mutate(class = factor(class, labels = c("NG", "OG", "TSG")))

# Box plots of each variable (7 per plot) split by class
for (i in seq(1, ncol(training0) - 2, by = 7)) {
  # Select 7 columns plus class (column 99)
  plot <- training0[, c(seq(i, length.out = 7), 99)] %>%
    # Scale and center numeric variables
    mutate_if(is.numeric, scale) %>%
    # Turns 7 columns into 2 columns: variable name and value
    gather(-class, key = "Variable", value = "Value") %>%
    ggplot(aes(x = Variable, y = Value, color = class)) +
      geom_boxplot(outlier.shape = 1) +
      theme(axis.text.x = element_text(angle = 20)) +
      coord_cartesian(ylim = c(-10, 10))
  print(plot)
}

### FUNCTION DEFINITIONS ###

removal_kfold_qda <- function(train, kfold) {
  # List to store average error rates in a list where each element represents
  # a removed variable.
  lst <- list(0)
  errors <- rep(lst, dim(train)[2] - 1)

  # Here we'll run 3 for loops to run k-fold of 5 and calculate
  # the mean error 100 times per variable to determine if removing
  # that variable imprives the model in any way.

  # The first loop with j as the index will be running through each variable
  # in our current training data
  for(j in 1:(dim(train)[2]-1)) {
    # Use "j" as index to remove a variable in each run
    rmv_sbst <- c(j)
    training_d <- train[, -rmv_sbst]

    # The second loop with k index will allow k-folds of 5
    # to run 15 times at each iteration.
    # At the end of each iteration, the average of the error rate
    # is calculated and stored
    for(k in 1:15) {
      # Empty vector to store errors from qda and lda models
      error_qda <- error_lda <- rep(0, 5)
      # k-fold CV, k = 5
      kfolds <- createFolds(training_d[, dim(training_d)[2]], k = kfold)

      # The third loop with i index will run the k-folds through each of the
      # 5 subsets created using the createFolds function. At the end of each
      # iteration, the average of the folds is taken and stored in error_qda
      for(i in 1:kfold) {
        # Here we subset the data by test and training sets
        # The current index will determine the current fold
        # of subsets for the test set
        testset <- training_d[kfolds[[i]], ]
        trainingset <- training_d[-kfolds[[i]], ]

        # Subset with only the response variable
        test_class <- testset$class

        # qda model found
        qda_m <- qda(class ~ ., data = trainingset)

        # Predictions
        qda_pb <- predict(qda_m, testset)

        # Calculation of error rate
        error_qda[i] <- mean(qda_pb$class != test_class)
      }
      # The average of the errors of each of the folds is stored here in errors
      errors[[j]][k] <- mean(error_qda)
    }
  }

  # We'll take the overall averages of the average error rates
  lapply(errors, mean)
}

addition_kfold_qda <- function(main_t, train, kfold) {
  lst <- list(0)
  errors2 <- rep(lst, dim(main_t)[2])

  # Here we'll run 3 for loops to run k-fold of 5 and calculate
  # the mean error 100 times per variable to determine if -adding-
  # a variable improves the model in any way.

  # The first loop with j as the index will run through each variable
  # in the set t1 in order to add that variable to our current training set
  for(j in 1:(dim(main_t)[2])) {
    # Use "j" as index to add a variable in each run
    add_sbst <- j
    training_d <- data.frame(main_t[, add_sbst], train)

    # Similarly to the loops where we remove a variable, this also runs
    # k-folds of five 15 times
    # At the end of each iteration, the average of the
    # 5 folds is taken and stored in errors
    for(k in 1:15) {
      # Empty vector to store errors from qda and lda models
      error_qda <- error_lda <- rep(0, 5)
      # k-fold CV, k = 5
      kfolds <- createFolds(training_d[, dim(training_d)[2]], k = kfold)
      # Also similar to the variable removal loops, this for loop
      # runs through each fold's subset and then calcaulates
      # the error rate at the end and stores it
      for(i in 1:kfold) {
        # Here we subset the data by test and training sets
        testset <- training_d[kfolds[[i]], ]
        trainingset <- training_d[-kfolds[[i]], ]

        # Subset with only the response variable
        test_class <- testset$class

        # Regression methods on first data set
        qda_m <- qda(class ~ ., data = trainingset)

        # Predictions
        qda_pb <- predict(qda_m, testset)

        # Calculation of error rate
        error_qda[i] <- mean(qda_pb$class != test_class)
      }
      # Storage of the mean of the average error rate of each whole k-fold loop
      errors2[[j]][k] <- mean(error_qda)
    }
    errors2[j]
  }

  names(errors2) <- names(main_t)
  lapply(errors2, mean)
}

### MODEL DEVELOPMENT ###

# We'll add new variables that can hopefully predict the classes better.
# THe variables were picked from the boxplots where I determine if they appear
# to be good predictors,
sbst2 <- c( 2,  4,  5,  8,  9, 13, 15, 18, 24, 26, 30, 32, 38, 39,
           42, 44, 45, 46, 47, 51, 55, 56, 57, 63, 66, 68, 69, 70,
           71, 72, 73, 74, 75, 76, 77, 78, 80, 93, 95, 96, 98, 99)
training3 <- training1[, sbst2]

set.seed(10)
error_qda <- error_lda <- rep(0, 5)
# k-fold CV, k = 5
```

```r
kfolds <- createFolds(training3$class, k = 5)

# Single run of 5-folds to calculate error rate of current model
for(i in 1:5) {
  # Here we subset the data by test and training sets
  testset <- training3[kfolds[[i]], ]
  trainingset <- training3[-kfolds[[i]], ]

  train_class <- trainingset$class
  test_class <- testset$class

  # Regression methods on first data set
  qda_m <- qda(class ~ ., data = trainingset)
  lda_m <- lda(class ~ ., data = trainingset)

  # Predictions
  qda_pb <- predict(qda_m, testset)
  lda_pb <- predict(lda_m, testset)

  # Calculation of error rate
  error_qda[i] <- mean(qda_pb$class != test_class)
  error_lda[i] <- mean(lda_pb$class != test_class)
}

# We now see a model with some strong potential, particularly with the qda model
error_qda
error_lda
mean(error_qda)

# We'll run some code where we'll remove variables from the current
# training set iteratively and determine which removed variable
# improves the model with a lower avaerage error rate

set.seed(8)

# Here we'll run removal_kfold_qda() to see if we should remove variables

a1 <- removal_kfold_qda(train = training3, kfold = 5)
a1 <- unlist(a1)
names(a1) <- names(training3[-dim(training3)[2]])
boxplot(
  a1, main = "Range of Error Rates - Removal", ylab = "Average Error Rates"
)
summary(a1)
```

```
print(c(which.min(a1), min(a1)))
head(sort(a1), 3)

# After running the for loop several times, we can see removing a few
# variables would improve the model by a bit

# Here I'll pick the 3 variables with the lowest error rates when
# said variables are removed
# For example: variable 28 (Broad_H3K4me2_percentage), when removed,
# gives an error rate of 0.08091007, an improvemnt over most other
# removed variables. Likewise, variable 39 (H3K79me2_height) and
# variable 32 (H3K4me1_height) drops the error rate.
# Now we remove those variables mentioned above.

sbst3 <- sbst2[-c(28, 32, 39)]
training3_5 <- training1[, sbst3]

error_qda <- error_lda <- rep(0, 5)
# k-fold CV, k = 5
kfolds <- createFolds(training3_5$class, k = 5)

# Single run of 5-folds to calculate error rate of current model
for(i in 1:5) {
  # Here we subset the data by test and training sets
  testset <- training3_5[kfolds[[i]], ]
  trainingset <- training3_5[-kfolds[[i]], ]

  test_class <- testset$class

  # Regression methods on first data set
  qda_m <- qda(class ~ ., data = trainingset)

  # Predictions
  qda_pb <- predict(qda_m, testset)

  # Calculation of error rate
  error_qda[i] <- mean(qda_pb$class != test_class)
  print(table("true class" = test_class, "predicted" = as.factor(qda_pb$class)))
}
error_qda
error_lda
mean(error_qda)
# Although not perfect, we can see an improvement using this confusion matrix
# table("true class" = test_class, "predicted" = as.factor(qda_pb£class))
```

```r
set.seed(777) # seed of 777 because that's a lucky number]
test4 <- testing[, sbst3]

qda_m <- qda(class ~ ., data = training3_5)

qda_p <- predict(qda_m, test4)

# Code used to create pred_version8_5
resultqda <- data.frame("id" = testing$id, "class" = qda_p$class)

# Now, I'll be adding one variable (from the main dataset) or
# removing one variable via for loops and depending on which method
# improves the error rate, I'll choose that.

# We create a subset from the original training set without the
# id and class variables, as well as
# without variables already being used by our current training data
t1 <- training1[, -c(1, sbst3, dim(training1)[2])]

# We remove these variables since they create a "rank deficiency" error
t2 <- t1[, -c(15, 24, 27, 30, 42)]

set.seed(777)

a2 <- addition_kfold_qda(main_t = t2, train = training3_5, kfold = 5)
# Create boxplot to see the range of average error rates
boxplot(
  unlist(a2),
  main = "Range of Error Rates - Addition",
  ylab = "Average Error Rates"
)
# Generate summary statistics about the average error rates
summary(unlist(a2))
# Shows the index of the minimum value as well as the value
print(c(which.min(unlist(a2)), min(unlist(a2))))
# Shoes the three most minimal values
head(sort(unlist(a2)), 3)

# We can see adding the variable MGAentropy improves the model the most

# We now add MGAentropy
sbst4 <- sort(c(36, sbst3))
training4 <- training1[, sbst4]
```

```r
# We now run another variable removal loop to test which variable we don't need

set.seed(1)
a3 <- removal_kfold_qda(training4, kfold = 5)
# Turns list of average error rates into a vector
a3 <- unlist(a3)
# Assigns the names of the variables that were removed to the correct place
names(a3) <- names(training4[, -dim(training4)[2]])
boxplot(
  a3, main = "Range of Error Rates - Removal", ylab = "Average Error Rates"
)
summary(a3)
print(c(which.min(a3), min(a3)))
head(sort(unlist(a3)), 3)

# Variable 17 (Super_Enhancer_percentage) can be removed

training4_5 <- training4[, -17]

set.seed(80)
test5 <- testing[, sbst4]
test5 <- test5[, -17]

qda_m <- qda(class ~ ., data = training4_5)

qda_p <- predict(qda_m, test5)

# Code used to create pred_version9
resultqda <- data.frame("id" = testing$id, "class" = qda_p$class)

# Now, I'll be removing one variable or adding one variable
# (from the main dataset) via for loops and depending on
# which method improves the error rate, I'll choose that.

t2 <- training1[, -c(1, sbst4, dim(training1)[2])]
# We remove these indices since they create a "rank deficiency" error
t2_5 <- t2[, -c(15, 23, 26, 29, 40)]

set.seed(7)
a4 <- addition_kfold_qda(main_t = t2_5, train = training4_5, kfold = 5)
boxplot(
  unlist(a4),
  main = "Range of Error Rates - Addition",
  ylab = "Average Error Rates"
)
summary(unlist(a4))
print(c(which.min(unlist(a4)), min(unlist(a4))))
head(sort(unlist(a4)), 3)

# We can add Missense_TO_Silent_Ratio

sbst5 <- sort(c(11, sbst4))

training5 <- training1[, sbst5]

test6 <- testing[, sbst5]

qda_m <- qda(class ~ ., data = training5)

qda_p <- predict(qda_m, test6)

# Code used to create pred_version10
resultqda <- data.frame("id" = testing$id, "class" = qda_p$class)

# We now run another variable removal loop to test which variable we don't need

set.seed(777)
a5 <- removal_kfold_qda(train = training5, kfold = 5)
a5 <- unlist(a5)
names(a5) <- names(training5[, -dim(training5)[2]])
boxplot(
  a5, main = "Range of Error Rates - Removal", ylab = "Average Error Rates"
)
summary(a5)
print(c(which.min(a5), min(a5)))
head(sort(a5), 3)

# We can remove the 28th variable as seen with the score

### FINAL MODEL ###

# We'll try out the model without the 28th variable now
# sbst5_5 <- c(2, 4, 5, 8, 9, 11, 13, 15, 18, 24, 26, 30, 32, 36,
#              38, 39, 42, 44, 45, 46, 47, 51, 55, 56, 57, 63, 66,
#              69, 71, 72, 73, 75, 76, 77, 78, 80, 93, 96, 98, 99)
sbst5_5 <- sbst5[-28]
training5_5 <- training1[, sbst5_5]
```

```r
test6_5 <- testing[, sbst5_5]

qda_m <- qda(class ~ ., data = training5_5)

qda_p <- predict(qda_m, test6_5)

resultqda <- data.frame("id" = testing$id, "class" = qda_p$class)

# All predictors used in best model
print(attr(qda_m$terms, "term.labels"))
```